SWDE - Software Development
Zusammenfassung FS 2019

Maurin D. Thalmann

19. Februar 2019

Inhaltsverzeichnis

1 Buildautomatisation

1.1 Sie kennen die Vorteile eines automatisierten Buildprozesses

- Automatisierter Ablauf, keine Interaktion mehr benötigt
- Reproduzierbare Ergebnisse
- lange Builds können auch über Nacht laufen
- Unabhängig von Entwicklungsumgebung

1.2 Sie können verschiedene Beispiele von Buildwerkzeugen benennen

Make (für C/C++ Projekte), Urvater der Build Tools,
hohe Flexibilität, gewöhnungsbedürftige Syntax

Ant Java mit XML

Maven Java mit XML

Buildr Ruby-Script

Gradle Groovy Script mit DSL

Bazel Java mit Python-like Scripts

1.3 Sie beherrschen die Anwendung eines ausgewählten Buildwerkzeuges (Apache Maven)

Beherrschen muss man es selber, es kann entweder aus der Shell (Terminal/Konsole) verwendet werden oder aus den integrierten Funktionen in der IDE selbst.

1.4 Sie sind mit den wesentlichen Konzepten von Apache Maven vertraut

Deklaration des Projektes in XML, zentrales Element pro Projekt ist das **Project Object Model (POM)**, welches Metainformationen, Plugins und Dependencies definiert. Basiert auf einem globalen, binären Repository. Plugins werden durch Dependencies dynamisch ins lokale Repository geladen ($HOME/.m2/repository)
Bei einem Buildprozess durchläuft ein Projekt einen Lifecycle mit folgenden Phasen:

validate validiert Projektdefinition

compile Kompiliert die Quellen

test Ausführen der Unit-Tests

package Packen der Distribution

verify Ausführen der Integrations-Tests

install Deployment im lokalen Repository

deploy Deployment im zentralen Repository

2 Versionskontrolle mit Git und GitLab

2.1 Sie kennen die Aufgaben eines Versionskontrollsystems und können grundlegend damit arbeiten

Grundlegende Arbeit:

checkout lokale Arbeitskopie eines Projekts erstellen

update Änderungen Dritter in Arbeitskopie aktualisieren

log Bearbeitungsgeschichte eines Artefakts ansehen

diff verschiedene Revisionen miteinander vergleichen

commit / checkin Artefakte ins Repository schreiben → aussagekräftiger Kommentar!

Tagging: Revisionsstand mit Namen markieren, Marke oder Version: 1.5.2beta o.ä. Nützlich bei Release eines Produkts (aber auch meilensteine, Testversionen, Auslieferungszustände, etc.) wird unterschiedlich realisiert.

Branching: Parallele, voneinander getrennte Entwicklungszweige (für Bugfixing, Prototypen, Tests, Experimente, nachvollziehbare Änderungsworkflows, etc.) Bei Nicht-Wegwerf-Entwicklungen später Merging möglich/notwendig.

Ausschliesslich Quell-Artefakte werden verwaltet, **NIE** generierte/erzeugt Artefakte einchecken, können mit Hilfe der SCM ignoriert werden (.gitignore)

2.2 Sie kennen die verschiedenen Konzepte und Arten von Versionskontrollsystemen

- Zentrale oder verteilte Systeme
- Optimistische und pessimistische Lockverfahren
- Versionierung auf Basis einer Datei, Verzeichnisstruktur oder der Änderung (changeset)
- Transaktionsunterstützung (vorhanden oder nicht)
- Verschiedene Zugriffsprotokolle und Sicherheitsmechanismen
- Integration in Webserver (vorhanden oder nicht)

2.3 Sie können mit verschiedenen (Client-)Werkzeugen von Versionskontrollsystemen alleine und im Team arbeiten

CVS UT-Versionskontrollsystem, stabil, wenig Fehler, einfache Anwendung, ABER nur dateibasierend, Verzeichnisstruktur nicht versioniert, unterscheidet zwischen Text- und Binärdateien, Ablage von Binärdateien platzintensiv, keine Transaktionen

Subversion Transaktionsorieniert, versioniert ganze Verzeichnisstruktur, optimierte/effiziente Speicherung und Übertragung, Repositorystruktur frei wählbar (für Experten flexibler, für Anfänger schwieriger), Integration in Webserver möglich, aber Branching und Tagging technisch eig. Kopien/Links

git verteiltes System, primär lokale Arbeit, beliebig viele Server/Repos möglich, auch rein lokal einsetzbar, skaliert, Integration mit zusätzlichen Web-Applikationen, erfordert aber ein solides Konzept, für Einsteiger schwierig, da sehr mächtig und viele Funktionen